Formal Modeling and Analysis of Autonomous Robotics System Using Z

Yujian Fu\textsuperscript{1}, Nithin Yama\textsuperscript{1}, and Zhijiang Dong\textsuperscript{2}
\textsuperscript{1}Department of Computer Science, Alabama A&M University, Normal, AL, USA
\textsuperscript{2}Department of Computer Science, Middle Tennessee State University, Murfreesboro, TN, USA

Abstract—In recent years, the mobile robotics systems have been developed and attracted more and more attentions because of its applications in various disciplines. The software development of mobile robotics is a key and complex process and it is required to be reliable and correct when placed in the unknown working environment. To illustrate how to design reliable and correct robotics systems, software design is the key concern in robotics systems. Based on the set theory and first order logic, Z notation is a formal approach widely used for the modeling of distributed and embedded system design. In this research, we presented a formal framework based on the Z notation to describe generic behaviors of robotics systems. A case study of field guard robot that uses the LEGO NXT toolkit was implemented in Java to validate the framework. In the mobile robot navigation system, a robot travels from a start state to a specified final state. To navigate the position of robot and objects, coordinate systems are used to specify and analyze the paths from start state to the final state. Using Z based framework, it is able to investigate and analyze the entire formal specification of the autonomous mobile system.

Keywords: Mobile robotics systems, software design, formal framework, Z notation

1. Introduction

Many applications of autonomous mobile robots are developed and widely used in various disciplines, such as, manufacturing, construction, waste management, underwater task, space exploration, medical surgery, serving and assistance for the disabled people. Autonomy and navigation are two major concerns in a broad domain covering a large spectrum of applications and technologies. They have drawn more and more researchers attentions due to multiple disciplines as well as the advanced techniques involved. Therefore, how to design a reliable and correct mobile robot system is a challenge issue now. Making progress toward autonomous robots is of major practical interest in a wide variety of applications.

It is a fundamental requirement for any autonomous robot that is able to navigate from one location to another. Avoiding the dangerous situation on the path such as collision with obstacles, falling in a cave to stay in the safe operational environment is the first class requirement during navigation. Navigation is an ability to identify robot’s current position, calculate the new path facing the obstacles while remember and travel towards the destination. To build reliable and robust autonomous robotics systems and achieve the quality of navigation, different approaches have been proposed. In this research, we use Z notation to model mobile robot including navigation. The problem scenario we examine is an autonomous robot that travel through an area that limited by color line. The robot needs to find all obstacles, collects them and removes to certain area and goes back to the place that obstacle was found. The mobile robot needs to be able to navigate within the area, identify current, collect locations and calculate the path between.

Formal methods are mathematical based notations that can precisely describe the system in an abstract level by reducing ambiguity and inconsistency. The Z notation is based on the set theory and first order logic with denotational semantics. However, due to the complexity of the symbols and knowledge, formal methods are not widely accepted in industry. The basic unit in Z notation is a schema, each data and operation can be specified by a schema or set of operations of schemas. This research work modularizes the Z notation into several blocks of robotics systems and expresses in the set of schemas. Therefore, it will facilitate further research in the automatic validation and verification tool.

The remainder of this paper is organized as follows. Section 2 gives a brief description of Z notations. In Section 3, we discuss the related works of formal modeling of robotics systems. After that, a Z model of robotics systems is presented in Section 4. A case study using LEGO NXT toolkit is presented in Section 5. Conclusion and future investigations are discussed in the Section 6.

2. Overview of Z Notations

Z notation is a formal specification notation that was first created by J. R. Abrial and further developed by the Programming Research Group at Oxford. It is based on the well-known mathematical concepts of set theory and predicate logic.

The primary construct in Z notation is called schema, which represents a block, an operation, a set of data or a subsystem. A schema is defined by two parts – data
declaration and system constraints. The data declaration consists of the definition of necessary data of the schema with data types in the form of \( \nu : T \), where \( \nu \) is the variable and \( T \) represents the type of data (data set). The constraints of the system can be specified by a set of predicates usually denoted by first order logic. In the operation schema, the constraints of the system includes the precondition and postcondition that define the relationships between the declared variables. A schema has following form:

```
Schema

//declaration

//preconditions

//postconditions
```

Data types in Z notation can be defined as basic or composite. The basic data types are usually denoted by capitalized letters. Composite data types can be the cartesian product and schema types. Any data defined must have a data type.

The operations are functions that maps a data set to another. The mapping function can be total or partial relation.

Any schemas in Z notation can be either state schemas or operation schemas. State schemas capture the static aspect of a system by defining the database, constraints and initial data. Operation schemas capture the dynamic aspects and define input-output relations. In other words, operation schema defines an operation in terms of the relationship between the state before the operation executes and the state after it has completed execution. The declaration part contains variables representing the before and after states, input and outputs. The constraints specify the precondition that can cause state change and postcondition after state change. For each system model, it is important to identify the key data and operations. Following notations are conventionally used in the content:

- Unprimed variable (e.g. \( \nu \)) – value of a variable before execution of an operation.
- Primed variable (e.g. \( \nu' \)) – value of a variable after execution of an operation.
- Variable suffix with ? – an input variable to an operation.
- Variable suffix with ! – an output variable to an operation.
- \( \Delta S \) – denotes change for the data in set (or database of) \( S \).
- \( \Xi S \) – denotes no change for the data in set (or database of) \( S \).

3. Related Work

There are a lot of works have been done on the modeling and analysis of embedded systems using Z notation, but only very few works that has been done in the modeling robotics systems using Z notation in the literature. In this section, we will overview the works that applied formal methods to the robotics systems.

A model to guide and keep track of a robot such that it is able to complete several tasks is described in [9]. Petri nets and Petri net extension methodologies have been used to model systems for the control and coordination in the unstructured environment. Murata et al. [7] presented an algorithm to construct predicate/transition models of robotic operations. Basically, robot actions were described as enabled and firing transitions and the model was used for the planning of concurrent activities of multiple robots. The model shows the ability of the Petri net models to capture interactions between the agents that are not evident in the design process. In a similar way, Xu et al. [12] proposed a methodology based on predicate/transition nets for multiple agents under static planning of activities. In addition, they proposed a validation algorithm for plans with parallel activities. The work of Leitao et al. [5], proposed a Petri net model approach to formal specification of holonic control systems for manufacturing. They developed a Petri net submodel for each of the four types of holons (agents) suggested in the ADACOR (Adaptive Holonic Control Architecture for Distributed Manufacturing Systems) architecture. There was no attempt to study the structural properties of the Petri net model in order to assess some sort of dependability in the proposed architecture. Now the existing navigation of single robot has extended to multiple robots system. The multiple robots addressing the issues of cooperation and formation control is discussed in several works ([3], [2]) where reactive behavior based approach to formation control is described. Some other works use Petri net plan can be found in [6-13].

Finite automata and graph theory [4] have proved to be useful mathematical models for robot navigation through a discrete environment. In [8], finite automata are used to control the navigation of mobile robot along the possible paths. They analyzed the movement and controlled the population of a mobile robot by applying the algorithms of graph theory. A number of modeling techniques for mobile robot have been developed by researchers such as partially observable Markov and behavior based navigation but the existing approaches have lack of the formalization. Melo, Isabel and Lima in [6] has examined the problem of multirobot navigation. They have analyzed the problem of driving a robot population moving in a discrete environment from some initial to a target configuration. In this paper, we build a modeling framework using Z notation to formal describe autonomous mobile robot systems.

4. A Modeling Framework of Robotics Using Z Notation

Formal specification language (FSL) is a mathematical based notation on top of algebraic, logics and/or discrete mathematics. Therefore, FSL can provide a precise analysis
and consistent reasoning on the system properties from design model. However, to establish a formal model is not a straightforward job because it is challenging and tricky to connect the mathematical notations to the dynamic and flexible real world systems.

One of the commonly used formal specification languages is Zed language or Z notation, which is based on the set theory and predicate logic. The Z notation specifies the system by input-output relation and describes the operations by the constraints on the states. Two typical components in the Z notation are state and operations that represent the data and events of the systems. The fundamental block in Z notation is schema that is composed of states and operations. In this case, Z notation provides a precise and simple style for the system model. The fundamental unit of the Z notation is a schema, which includes two parts – data and constraints. There are two types of schemas – state schema and operation schemas. In the state schema, the data of the system and initial conditions of the data are specified. In the operational schemas, the data that is affected and operated on, the pre-conditions and post conditions are described as set notation, predefined syntax and first order logic. For specific syntax of the Z notation, readers can refer to Spivey’s book [10].

The modeling framework of autonomous robot includes three major components – environment, controller, sensor. The environment schema aims at description of the scope. Two types of approaches can be used – coordinate systems or directed graph (DG). In this work, we present a model based on the coordinate system. Assume a two dimensional system, a coordinate system is described by a triple $O = <O, X, Y>$, where $O$ is the origin, and $X$ and $Y$ are horizontal and vertical numbers respectively.

$$\begin{align*}
\text{Environment} \\
CS : ORIGIN \times HX \times VY \times RC \times OC \\
\forall c \in CS. c[1] == <0,0> \land c[4] \neq \lambda
\end{align*}$$

In the above schema of Environment, any data is specified by five fields – origin, horizontal number, vertical number, robot position and obstacle position. It requires that the origin starts at $<0,0>$, and robot is within the scope and should be known $c[4] \neq \lambda$. The coordinate system provides the direction for the robot movement. Similarly, we can define the schema of Robot as follows:

$$\begin{align*}
\text{Robot} \\
Robot : ORIGIN \times HX \times VY \times STATUS \times OC \\
\forall r \in Robot, e \in Environment. r[2] == <0,0> \land c[4] \neq \lambda
\end{align*}$$

The $Robot$ state schema is defined by a data with five fields. The first field specifies the original coordinates of the robot, second and third place specify the current location, while the last filed specifies the robot status, which includes $forward$, $backward$, $moving$, $spinning$, $turnleft$, $turnright$, and $Uturn$.

Now we can define sensor component based on the above two schemas. The sensor component includes several schemas of the various sensors. The schema of sensor highly depends on the sensing mechanism. In this paper, we define ultrasonic sensor and light sensor based on the LEGO NXT Mindstorm toolkit as follows.

$$\begin{align*}
\text{UltrasonicSensor} \\
\text{UltraSensor : FREQUENCY} \times \text{DIST} \\
\forall u \in \text{UltraSensor}. u[2] \leq 1.5 \land u[1] \neq \lambda
\end{align*}$$

In the above schema of UltrasonicSensor, if there is an obstacle within 1.5m, then there is frequency returned and the obstacle can be detected.

$$\begin{align*}
\text{LightSensor} \\
\text{LightSensor : FREQUENCY} \times \text{COLOR} \\
\forall l_1, l_2 \in \text{LightSensor}. l_1[1] \neq l_2[1] \land l_1[2] \neq l_2[2]
\end{align*}$$

In the schema of LightSensor, it is expected that for all different colors the light sensor can return different frequencies. The schema of Sensor is a composite scheme and can be defined as

Sensor $\equiv$ UltraSensor $\oplus$ LightSensor

A path of a robot can be defined as the distance from source to destination. A path needs to be updated every time interval due to the environment changes. The robot needs to remember the recent path. The calculation of path is the line between two points – current position and the destination. Considering one obstacle between the current position and the destination, a path is two coordinates for robot and destination.

$$\begin{align*}
\text{Path} \\
\text{Path : HX} \times \text{VY} \times \text{HX} \times \text{VY} \\
\forall p \in \text{Path}, e_1, e_2 \in \text{Environment.} \\
\end{align*}$$

The controller schema can be complicated and highly depends on the requirements and tasks of the robot. The controller takes input from sensor, perform some tasks and output the commands to actuators to drive the robot. The task can be calculation of the shortest path, uploading some devices to help movement (e.g. a pad to the foot so that the robot can walk cross the river.), and/or starting other motors (rescue people). In this framework, we only present the shortest path calculation so that the robot can avoid the collision with obstacle and move to the destination.
The system needs current robot position ($rc?$) and obstacle position ($oc?$ $\leftarrow x, y$). In addition, the frequency of the ultrasonic sensor ($freq?$) is required for the sensor to return the knowledge of detected object and ($des?$) denotes the destination position that is retrieved from memory. The precondition for the robot to calculate the correct moving path is the detection of obstacle is precisely evaluated. The path includes two parts – $path1$ represents the distance from the robot to the obstacle, while $path2$ describe the distance from obstacle to the destination ($des?$).

The preconditions for the controller are: i) the robot, obstacle and the destination are all identified in the coordinate systems ($Environment$); ii) the value returns from the sensor matches the value for the obstacle. After the controller calculates the path, the new paths should be generated, commands are sent to motors to turn in a certain angle and move the robot.

\[
\begin{align*}
\Delta & \text{Environment} \\
\Delta & \text{Robot} \\
\Delta & \text{Path} \\
\Xi & \text{Sensor} \\
rc? & \in \text{Environment} \\
oc? & \in \text{Environment} \\
ds? & \in \text{Environment} \\
msg! & : \text{MSG} \\
\end{align*}
\]

\[
\forall e_1, e_2, e_3 \in \text{Environment}, \\
\land e_3[3] == des?[3] \\
\text{Environment'} = \text{Environment} \cup \langle e \rangle \\
\langle e_1, e_2, e_3, rc?, oc? \rangle \\
\text{Robot'} = \text{Robot} \setminus \{r\} \cup \langle rc?[1], rc?[2], rc?[3], \\
rc?[4] = \text{"moving"}, oc? \rangle \\
\text{Path'} = \text{Path} \setminus \{p\} \cup \langle rc?[2], rc?[3], oc?[2], oc?[3] \rangle
\]

The block diagram that describes the architecture of the framework is shown in the Figure 1. Fig. 1 show a generic view and summarizes the relations of the above well defined schemas.

5. Case Study Using LEGO Toolkit

The Mindstorms NXT brick uses a 32-bit ARM processor as its main processor, with 256 kilobytes of flash memory available for program storage and 64 kilobytes of RAM for data storage during program execution. To acquire data from the input sensors, another processor is included that has 4 kilobytes of flash memory and 512 bytes of RAM. Two motors can be synchronized as a drive unit. To give the robot the ability to “see,” the ultrasonic sensor, which is accurate to 3 centimeters and can measure up to 255 centimeters, and the light sensor, which can distinguish between light and dark, can be attached to the brick. A sound sensor that can be adjusted to the sensitivity of the human ear can be used to give the robot the ability to hear and react, if programmed, to noises. Finally, the two touch sensors give the ability for a robot to determine if it has been pressed, released, or bumped, and react accordingly [1].

As a replacement for the standard Lego firmware, the LeJOS project has support for threading, arrays, recursion, synchronization, exceptions, non-generic data structures, standard data types, and input and output [11]. The LeJOS virtual machine supports much of the java.util package, but the data structures require that data be stored as type Object and then cast to a type that inherits Object. For input and output, both streams and sockets are available for use. For control purposes, the LeJOS platform supports the direct connection of Bluetooth-enabled GPS units for spatial location information and keyboards for the navigational control of a robot [11].

5.1 Field Guard Robot (FGR)

LEGO NXT Mindstorm is a highly integrated toolkit for the robot educators and learners in different levels. The main features are i) is easy to assembly and ii) supports multiple interfaces and platforms for various levels of users. Java is supported by LEGO NXT toolkit by a plug-in package named LeJOS, a source forge project created to develop a technological infrastructure for LEGO Mindstorms products. LEGO NXT has three output ports to drive motors, and four input ports for data acquiring from sensors. To validate and apply the above formal framework, a LEGO NXT field guard robot (FGR) was assembled to search the objects in a certain field by patrolling around. The robot is going to make a buzz for six times and starts spinning in the center of the guarded area continuously looking for an object within the specified area around it; If it is found any object within the area, it will take and push the object to a certain area outside the guarded place. After that it comes back to the center of the area and continue to spin.

![Fig. 1: The Z Notation Framework for Autonomous Mobile Systems](image)
5.2 Modeling of FGR in Z

To ensure the system works correctly with expected behaviors, we apply the above Z notation framework on this LEGO NXT Mindstorm field guard robot (FGR). The main schemas include followings – Environment, Robot, UltrasonicSensor, TouchSensor, Pickup and Drop. Since we use the coordinate system for the robot navigation, the schema of Environment, Robot and sensors (UltrasonicSensor and LightSensor) are same as those defined in section 4. In this section a new sensor TouchSensor and some operation schemas will be discussed in this section. The touch sensor is used to react to the obstacle on the moving path. It responds only to the surface interaction with the object, and can be defined as following:

\[
\text{TouchSensor} \; : \; \text{VALUE} \times \text{STATUS}
\]

\[
\forall t \in \text{TouchSensor} \; t_1[1] == \lambda \land t_2[2] == \text{false}
\]

The data field for the touch sensor are the value read from touch sensor and the status of object existence. The status is a boolean value, where true indicates that the object is detected, otherwise, no object. The constraint of touch sensor is that if there is no touch with object, the status returned from touch sensor is false, which indicates no object in front of the sensor.

Two basic operations of the FGR can be pick up and drop the object on the path. Besides, we need to define the moving operation to instruct the robot move correctly. For the space limitation, we only show the pick and drop operation schemas, which are defined as follows:

\[
\text{Pickup} \; : \; \text{Robot} \times \text{Controller} \times \text{Sensor} \times \text{Environment} \rightarrow \text{Environment}, \text{and}
\]

\[
\text{Drop} \; : \; \text{Robot} \times \text{Controller} \times \text{Sensor} \times \text{Environment} \rightarrow \text{Environment}
\]

The initialized Robot schema can be:

\[
\text{Robot_Init}
\]

\[
\Delta \text{Robot}
\]

\[
\Xi \text{Environment}
\]

\[
<<5,5>,5,5,start,\lambda>
\]

Considering the success and failure of each operation, the pickup and drop operation can be defined as composite schemas, where

\[
\text{Pickup} \; : \; \text{Pickup_Success} \lor \text{Pickup_Failure}, \text{and}
\]

\[
\text{Drop} \; : \; \text{Drop_Success} \lor \text{Drop_Failure}
\]

There are two preconditions specified in the schema of Pickup_Success: i) the robot, object and destination are all in the environment (identified by the coordinate system) and ii) the touch sensor needs to interact with the object when pick up. After pick up the object, the path needs to be updated immediately. Similarly, if this operation is failed, it can be defined as following schemas:

In the drop operation, to simplify the case, let assume there is no obstacle on the moving path. Thus, we do not need to update the path from the obstacle to destination. Otherwise, another schema for moving needs to be defined. The drop schema can be defined as following.

\[
\text{Pickup_Success}
\]

\[
\Delta \text{Robot}
\]

\[
\Delta \text{Environment}
\]

\[
\Delta \text{Path}
\]

\[
\Xi \text{TouchSensor}
\]

\[
\text{rc} : \text{ROBOT}
\]

\[
\text{oc} : \text{ENVIRONMENT}
\]

\[
\text{des} : \text{ENVIRONMENT}
\]

\[
\text{value} : \text{VALUE}
\]

\[
\text{msg!} : \text{MSG}
\]

\[
\forall e_1,e_2,e_3 \in \text{Environment}, \exists r \in \text{Robot}.
\]

\[
e_1[2] == \text{rc}?[2] \land e_1[3] == \text{rc}?[3]
\]

\[
\]

\[
\]

\[
\forall s \in \text{TouchSensor} ; s[1] == \text{value}?
\]

\[
\text{Robot} = \text{Robot} \rightarrow \{r\} \cup \{<r[1],r[2],r[3],r[4]\}
\]

\[
\text{Environment} = \text{Environment} \rightarrow \{e\} \cup \{<e[1],e[2],e[3],e[4]\}
\]

\[
\text{Path} = \text{Path} \rightarrow \{p\} \cup \{<r[2],r[3],oc?[2],oc?[3]>\}
\]

\[
\text{msg!} = \text{"Objectpickedup"}!
\]

\[
\text{Pickup_Failure}
\]

\[
\Xi \text{Robot}
\]

\[
\Xi \text{Environment}
\]

\[
\Xi \text{Path}
\]

\[
\Xi \text{TouchSensor}
\]

\[
\text{rc} : \text{ROBOT}
\]

\[
\text{oc} : \text{ENVIRONMENT}
\]

\[
\text{des} : \text{ENVIRONMENT}
\]

\[
\text{value} : \text{VALUE}
\]

\[
\text{msg!} : \text{MSG}
\]

\[
\forall e_1,e_2,e_3 \in \text{Environment}, \exists r \in \text{Robot}.
\]

\[
e_1[2] \neq \text{rc}?[2] \land e_1[3] \neq \text{rc}?[3]
\]

\[
\land(e_2[2] \neq \text{oc}?[2] \land e_2[2] \text{negoc}?[3] \lor
\]

\[
\]

\[
\forall s \in \text{TouchSensor} ; s[1] \neq \text{value}?
\]

\[
\text{msg!} = \text{"NoObject"}
\]

The reason that the Path data was not updated because the Drop schema only takes care from the found obstacle to the destination where the path had been defined in the NXT controller. In this design, LEGO robot uses light sensor to identify the destination place by placing a certain color of
Next, we define the failure case of drop operation.

**Drop_Success**

- \( \Delta \text{Robot} \)
- \( \Delta \text{Environment} \)
- \( \Xi \text{Path} \)
- \( \Xi \text{LightSensor} \)
- \( r.c.? : \text{ROBOT} \)
- \( d.e.? : \text{Environment} \)
- \( \text{status}? : \text{STATUS} \)
- \( \text{value}? : \text{VALUE} \)
- \( \text{msg}! : \text{MSG} \)

\[ \forall e_1, e_2 \in \text{Environment}, \exists r \in \text{Robot}. \]
\[ (e_1[2] == r.c?[2] \land e_1[3] == r.c?[3]) \land \]
\[ \forall s \in \text{LightSensor}.s[1] == \text{value}? \]
\[ \forall r \in \text{Robot}.r[5] = \text{status}? \]
\[ \text{Robot}' = \text{Robot} - \{r\} \cup \{<r[1], r[2], r[3], r[4] = \text{"drop"}, r[5] = d.e?[3]\} \]
\[ \text{Environment}' = \text{Environment} - \{e_1\} - \{e_2\} \cup \{<e[1], e[2], e[3], e[4] = d.e?[3], e[5] = \lambda >\} \]
\[ \text{msg}! = \text{"Object is dropped!"} \]

**Drop_Failure**

- \( \Delta \text{Robot} \)
- \( \Xi \text{Environment} \)
- \( \Xi \text{LightSensor} \)
- \( r.c.? : \text{ROBOT} \)
- \( d.e.? : \text{Environment} \)
- \( \text{status}? : \text{DOUBLE} \)
- \( \text{value}? : \text{VALUE} \)
- \( \text{msg}! : \text{MSG} \)

\[ \forall e_1, e_2 \in \text{Environment}, \exists r \in \text{Robot}. \]
\[ (e_1[2] \neq r.c?[2] \land e_1[3] \neq r.c?[3]) \lor \]
\[ \forall s \in \text{LightSensor}.s[1] \neq \text{value}? \]
\[ \forall r \in \text{Robot}.r[4] \neq \text{status}? \]
\[ \text{msg}! = \text{"No object is dropped!"} \]

There are several conditions that drop operation can be failed. For example, the robot is not in the coordinate system (not in the environment defined); the destination is not in the coordinate system, the sensor cannot detect the destination (by color), or the robot does not move (status is not properly set up to the correct value). There is one case that the robot can fail and not defined in the above schema – object is not hold in the claws. It is mostly caused by hardware imprecision not software controller based on our observation.

### 5.3 Implementation and Discussion of LEGO FGR

The LEGO FGR was implemented in Java. The set up includes the Eclipse IDE, LeJOS plug in for the APIs defined for LEGO robot accessories and conversion of from Java code to NXT brick. The java implementation of the LEGO FGR can realize all defined functionalities and finish expected requirements.

From the implementation of LEGO NXT robot in Java, we found that:

1) The model of robot is very important for the description and precisely implement the functions. There are some cases was missed in the code but defined in the model during implementation.

2) On the other side, it is noticed that the above model is a formal framework for the robot design. Due to limitation, the coordinate system and robot may need more detail description.

3) LEGO is a highly integrated tool kit. Some data fields have been specified properly without definition. A more sophisticated setting is needed for the more research study.

### 6. Conclusions and Future Works

We have presented a frame of Z notation for modeling the dominant design paradigms used in autonomous mobile systems. We have done this by taking advantage of formal specification languages to allow for navigation system, network connectivity and proactive process migration. We have also used the set theory and first order logics our models. In addition, the modular approach is used for the framework so that the robotics system with navigation can be modeled incrementally, exploiting commonalities among the design paradigms and reusing the defined state schemas and improving the system maintenance. Modularity can similarly be used to extend our models to represent their realization with specific technologies, or to capture their use in specific applications.

The formal basis of Z notation allows us to use the models and their potential extensions to reason about properties of mobile systems. This process can be aided by taking advantage of the substantial reuse of classes within our framework, and the particular form to which our specifications conform. Future work will look at how these aspects of the models can be exploited to simplify reasoning and, hence, the development of suitable reasoning support tools.

### Acknowledgment

The authors would like to thank all reviewers for the kindly comments and suggestions on this work.
References